JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

Model-based Testing Approaches: A Survey

Nidaa Ghalib Ali[1], Iman Mohammed Burhan[2], Keyvan Mohebbi[3, 4,]
Corresponding Author

[1] Technical Institute of Babylon, Al-Furat Al-Awsat Technical University (ATU), Iraq.
[2] Department of Computer Science, College of Medicine, University of Babylon, Iraq.
[3] Department of Electrical and Computer Engineering, Mobarakeh Branch, Islamic Azad University, Isfahan, Iran
[4] Department of Computer Engineering, Isfahan (Khorasgan) Branch, Islamic Azad University, Isfahan, Iran
Correspondence email: k.mohebbi@mau.ac.ir

ABSTRACT
Background: Model-based Testing (MBT) is a software testing technique in which the product under test's run-time behavior is compared to model predictions throughout the testing process. Representation of a system's behavior in a model. Actions, conditions, input sequences, output, and data flow from output and input can all be applied to represent behavior. It should be understandable and reusable in practice; the system under test must be described in detail to be shared. This paper studies the preliminaries of MBT. Then, it reviews the prominent approaches in this area and performs a comparative evaluation of the related works. The pros, cons, and open issues of MBT are also investigated.
Keywords: Model-based Testing, Survey, Open Issues, Pros and Cons

1. Introduction

Testing is an important but expensive activity in the software development life cycle. With advancements in the model-based approaches for software development, new ways have been explored to generate test cases from existing software models of the system, while cutting the cost of testing at the same time. These new approaches are usually referred to as model-based testing [1]. A software model is a specification of the system which is developed from the given requirements early in the development cycle [2] [3]. In the model-based development, this model is then refined until a required abstraction level is reached from which the implementation code can be generated or written by hand. MBT is an approach for deriving tests from the models using automated techniques [4]. The executable test suite can connect with the system being tested directly. Abstract test cases can be converted into concrete ones that can be executed. To build executable test suites in various MBT environments, models include enough information [5]. To produce a concrete test suite, items from the abstract test suite must be translated to specific techniques in the software. The mapping problem has been solved. In the case of online testing, abstract test suites exist only conceptually, not as tangible objects [6]. Models can be used to generate tests in a variety of ways. There is no recognized single optimum strategy for test derivation because testing is frequently experimental and reliant on heuristics. Test requirements, even use case(s), or test purpose are popular terms for a single package that contains all test derivation-related parameters [7] [8]. Test stopping criteria (also known as focus criteria) or information about which elements of a model should be tested can both be included in this package [9]. MBT is often seen as a type of testing of black box models rather than source code used to create test suites. MBT is still a developing field for large software systems. Let's look at the most basic case of modeling an application. Assume you're on a web application's login page [10]. Either log in (the first state) or go to "lost password" (the second stage) or reset your password as an individual user (third state). The user can return to the initial state from the second and third states. The state transition diagram can be used to model this flow or series. The basic login scenario can be modeled in this way. Figure 1 shows an example of the interconnected model for an application. Similar to user and data flow combinations, there could be many models for the overall application [11]. The application's numerous models are then linked together. MBT frameworks are divided into two categories.

1. Offline / a priori: Preparation of Test Suites before execution. A set of test cases constitutes a test suite.
2. Online / on-the-fly: During the test execution, test suites are created.

The objectives of this paper are to know the exact meaning of the MBT, the applications and related works of MBT, the difference between the related works of MBT, the benefits, and the wave of future for MBT.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

The organization of the paper is as follows. Section 2 introduces the research methodology. Section 3 reviews the related works. Section 4 discusses the pros and cons of MBT. Section 5 includes the open issues of MBT. Finally, section 6 concludes the paper.

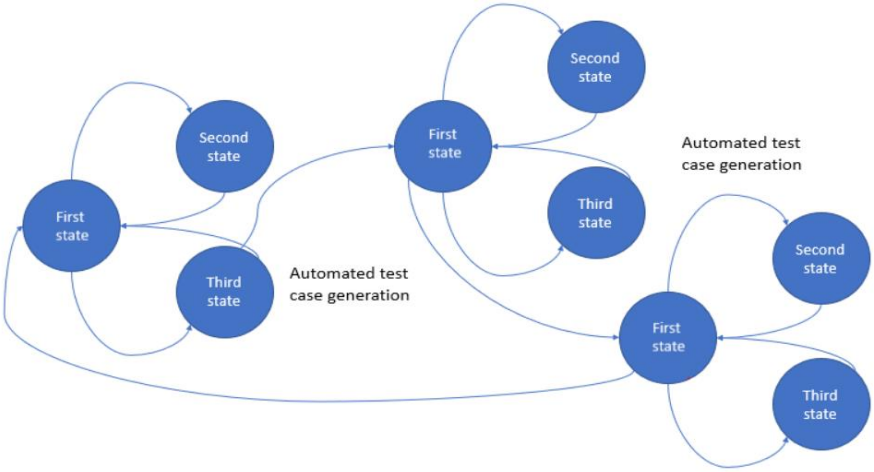

Figure 1. Interconnected model of an application [11]

2. Research Methodology

To address a research topic, the reviewer first outlines a review methodology that will be followed throughout the process. To ensure that our review adheres to principles for systematic literature reviews in software engineering, our protocol clearly outlines the search process, research question scope of review, exclusion/ inclusion factor of discovered tools, and factors of evaluation. Our research focused on MBT tools, but we also examined research literature to gain a deeper understanding of MBT, specifically state-based MBT. That's why our team of experts went through an extensive literature review to determine the criteria for selecting and comparing the investigated MBT tools as well as the criteria for excluding or including them in the so-called primary studies. The systematic review protocol includes these criteria [12].

This was the first step in our investigation, and we used a combination of eight different online databases as well as Google Scholar and the IEEE Explorer, ACM Digital Library as well as other sources like Elsevier and Springer Link. We also used Inter-Science Wiley and EI Engineering Village [13] [14]. We conducted our initial search in four distinct methods since we were looking for commercial MBT technologies that aren't often reported in academic, peer-reviewed journals. As a starting point, we looked at several MBT-related books and documents (articles, reports) available online. We also consulted a huge MBT discussion group personally, and a list of mailing for MBT-interested software testers. Our research methodology is summarized below: 1. to conduct electronic searches, and identify search strings to see if any of these abstracts are relevant to our work. 2. Collect technical reports, tool user manuals, white papers, books, video tutorials, case studies, presentations, demonstrations, and descriptions of the product, feedback from customers available in soft and hard formats on the internet and vendor website. 3. To get further information, look through a list of paper authors, references, and/or the websites of research groups and initiatives. 4. To increase your grasp of MBT tools, start and engage in discussions of technical online. 5. We kept going back to stages 1-5 until there was nothing else, we could add to the list of MBT tools and supporting data. Each model depicts a particular component of the system's behavior, and there are several to choose from. The model includes the following examples: Control Flow, Data Flow, Decision Tables, Dependency Graphs, and State transition machines. MBT explains how a system responds to an activity in terms of its behavior (described by a model). Supply the action and watch to see if the system responds as expected. Formal methods for verifying a system are light in weight. This testing method can be used to test both software and hardware. It is possible to derive test cases from a formal model based on a software specification using MBT. MBT is made up of numerous activities, as shown in Figure 1. User inputs requirements or specifications into a model of expected system behavior, which the test creation tool can then utilize to generate test cases. Graphical, textual, or diagrammatic, such as UML diagrams and finite state machine formalisms, can be used for this format. The high-level test objectives are represented by the formal model, which expresses the characteristics of behavior for a system that the engineer needs to evaluate. How large a test is determined by the coverage criteria that the tool employs to build tests? Which sections of the model are tested, how long does it take to produce them, and how long does it take

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

to develop them? Test generation criteria can be divided into "family" groups, such as (i) Control-flow through programs is handled by structural model coverage criteria, which are based on concepts from control-flow via the model; (ii) criteria for ensuring that the input data space for a transition or operation in the model is adequately covered; (iii) criterion for generating test suites that are good in detecting specific faults in the model; (iv) Criteria for generating a test suite that ensures the testing of all informal requirements; (v) Allows a validation engineer to specify that the model should be used to construct a specific test or set of tests; and (vi) generation methods of statistical test, which make use of random generation as a simple way to construct tests that look at a wide range of system characteristics. The test generation produces a test suite, which is a collection of abstract test cases that are sequences of traces from the model. All conceivable use cases are represented by a trace. An alternate step before returning the test cases is to choose a test selection heuristic. Since the generation technique might yield a significant number of test cases, the tester's work can become impractical if they don't do this. This work uses selection heuristics to decrease the size of the test suite by either rejecting comparable test cases using a similarity degree function or allowing the user to choose which scenarios to test. The abstract test cases are then exported and concretized into executable and/or human-readable formats as part of the MBT process. Often, this occurs through the use of translation or transformation software. In the meantime, this project doesn't support this phase. Finally, a test running is used to conduct the test. Abstract test cases are transformed into precise manual plans and test procedures for manual test implementation in the case of manual execution [16] [17] [18]. Figure 2 shows the MBT tool for statistical and functional testing.

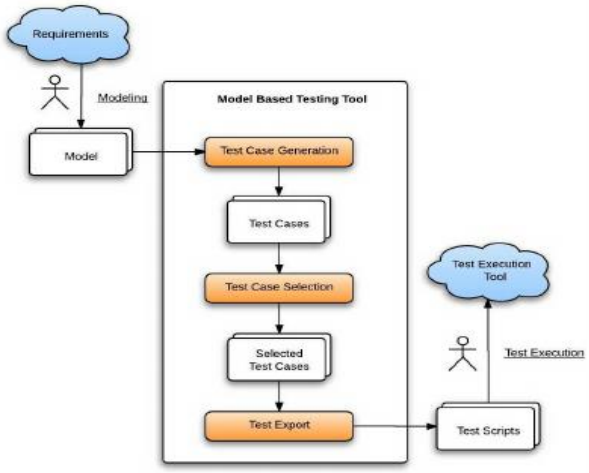

Figure 2. MBT Tool for Statistical and Functional Testing

3. Literature Review

Comparison of manual, semi-automatic, and fully automated testing, and MBT has been offered by several researchers [19], [20], [21] to determine the MBT's time and time effectiveness. We also discovered two papers that examined the effectiveness of Graph Walker (GW) in terms of fault identification and cost [22] [23] [24]. Summaries of the relevant research are provided in the next part, although the reader is referred to various review publications on MBT for more exploration of the topic [25] [26] [27].

In, a noteworthy comparison for testing a web-based application using MBT was offered [28]. MBT is a systematic method that can uncover more functional defects than manual testing, according to the findings. Nevertheless, activities of MBT took more time than manual test design. It was also discovered that manual test design was more effective in finding GUI-related errors. [15] also provides an overview of a novel MBT model, as well as a comparison of manual and automated test creation methods. By mapping it with abstract tests developed applying MBT, the authors offered to test many applications language for the development of concrete test values. They also provided an empirical assessment of the mapping between abstract and concrete tests generated by automated vs. manual means. Automatic test production gives efficient tests with superior mapping, according to the empirical evaluation.

In the networking sector, Pretschner et al. [29] did an empirical investigation to compare the manual and automatic MBT methodologies in terms of model coverage, discovered defects, and code coverage. The goal of this research was to assess the quality of created tests using both methods and to investigate the relationship between model and running

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

coverage. A moderate association was found between the deployments of a decision model. In addition, the study found that MBT is more effective in detecting logical and requirement flaws [29]. Marques et al. [18] used MBT to compare manual and ad-hoc testing methodologies. The study evaluated the effectiveness, precision, efficiency, and relative recall of strategies for bug discovery. The findings revealed that the strategies for bug discovery are nearly equal in terms of time, precision, effectiveness, and relative recall. Mathematical analysis of the comparison between the two strategies revealed that both techniques have an essentially identical influence on bug detection.

Mouchawrab et al. [30] for the detection rat for state-based round-trip path coverage and structure-based edge coverage tests, factors that affect the performance of both techniques, and cost were evaluated from the tester's perspective. The results of the empirical investigation demonstrated that structural and state-based testing may be applied to improve fault identification rates. Similarly, Enoiu et al. [31] analyzed the effectiveness and efficiency of human and automated tests for defect detection based on the specifications and implementations of the tests. Specification-based tests, according to the findings, are more effective at detecting defects than implementation-based automated tests. Implementation-based tests, on the other hand, have a greater fault detection efficiency rate than manual specification-based testing.

Rashid et al. [32] a tool (CANoe+) utilizing GW and CANoe to rise the coverage of functional of produced test cases and comparing its effectiveness in fault detection with CANoe was built and assessed. According to the findings, CANoe+ enhanced function coverage by reporting all failed functionality, whereas CANoe did not report a single injected error. Similarly, an exploratory study in [33] employed GW to explore the cost-effectiveness of a keyword and behavior-driven test automation system in Agile Development (AD). According to the findings, MBT increased AD's maintenance, functional coverage, and flexibility, while reducing the cost of integration.

Overall, additional knowledge of how to efficiently construct models utilizing model-based test cases and how they compare to manual test cases based on many sources of information in industry is needed to evaluate against the relevant industrial systems, and various MBT strategies [34]. Figure 3 shows the model-based testing of distributed systems.

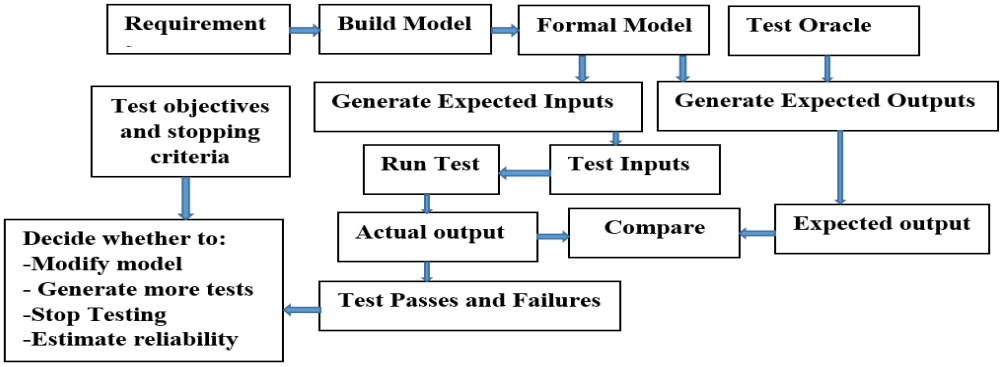

Figure 3. Model-based testing of distributed systems [35]

Table 1 shows a summary of the comparison of related works.

Table 1. Comparison of Related Works

References	Generation	Specification	Test type, Implementation, Observability	Test Selection Criteria
[35]	Previous Execution	Behavior, dynamic data flow graph	Integration tests, object-oriented, allows black box	Change of code

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

[36]	None	Behavior, dynamic control flow graph	Integration tests, allows black box	Change of code
[37]	None	Behavior, dynamic program slice	Integration tests, object-oriented, allows black box	Change of code
[38]	None	Behavior, dynamic call graph	Integration tests, allows black box	Change of code
[39]	None	Behavior, program dependency graph	Integration tests	Change of code
[40]	None	Behavior, call graph	Integration test, system test, object-oriented, white box	Change of specification
[41]	None	Behavior, extended finite state machine	Integration test, procedural, white box	Change of specification
[42]	Previous execution	Structure and behavior, UML sequence diagram, class diagram	Integration test, system test, object-oriented, white box	Change of specification
[43]	Previous execution	Structure and behavior, specification description language	Integration test, system test, object-oriented, white box	Change of specification
[44]	Previous execution	Structure and behavior, component dependency model, UML sequence diagram and directed graph	Unit test, system test, procedural, white box	Change of specification
[45]	Previous execution	Structure, semiformal (UML class diagram)	Unit test, system test, procedural, white box	Change of specification
[46]	Offline, simulation: UML SC for PLC	Target variable chosen manually, mutant created accordingly, every path checked with	Five mutation operators according to target	Mutation operators

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

		every mutant		
[47]	None	Timing sequence diagram	Fault operators (missing signal)	Path coverage, possible inputs, code splicing, possible paths leading to component
[48]	None	Fault operators	C, FBD, integration	Fault operators (missing signal)
[49]	None	Mathematical description of functions and safety properties	real system, IEC 61131-3, system	As spec.
[50]	None	Selection of component/ fault nodes in simulation model	SiL, IEC 61131-3, unit	As spec.
[51]	None	Fault models, failure mode function (mutants and saboteurs), requirements (Simulink assertion blocks), time window	MiL	As spec.
[52]	None	Fault model (saboteurs, mutants), some mutants generated automatically	C (generated out of SCADE), unit	As spec.
[53]	None	Stuck-at, open, complex logical faults,	Matlab/Simulink, unit	As spec.

4. Pros and Cons of Model-based Testing

Lorem There are benefits and drawbacks to every type of testing. When you're performing model-based testing. When adopting any type of testing, it's crucial to keep in mind that the initial investment in the testing can be substantial [54]. Model-based testing can take a lot of time upfront, but the ease with which testing can be automated will well outweigh

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 2879 - 2889
https://publishoa.com
ISSN: 1309-3452

the time invested. Issues will be addressed early in the development cycle so that the product does not enter the market with a slew of problems that detract from the consumer experience [55].

Model-based testing can be confusing for teams that are just getting started, so they may wish to hire testing professionals who can explain the approach to developers and help them put it in place. While hiring these professionals upfront may cost a little more money, it will save you money in the long run. Organizations seeking to achieve the maximum possible testing coverage might benefit greatly from the combination of cheap maintenance costs and high coverage. This type of testing will necessitate the presence of qualified testers within the company. Automated testing can, of course, be used in a variety of ways to ensure the greatest level of quality assurance [56] [57].

1. While the initial implementation of different test cases is arduous, once they are in place, maintaining this test is considerably easier.
2. Automated testing considerably reduces the costs of conducting many tests.
3. Tests can be run on several workstations or servers to see how the underlying system reacts to the various tests that are being run.
4. A system or product defect can be rapidly identified and corrected.
5. To make projects function more efficiently and smoothly.
6. Quality testing saves time and improves customer satisfaction.
7. When MBT testing is in place, testers who are testing multiple methodologies will discover that job satisfaction is significantly higher.

Cons of model-based testing include:

1. MBT necessitates teams of experienced testers.
2. The project's overall testing expenses will rise, though these costs are normally mitigated as the product matures.
3. Implementing any type of testing, but notably, MBT testing has a steep learning curve.

MBT testing can be difficult to comprehend at first, thus there is a learning curve that must be overcome

5. Open Issues of Model-Based Testing

Many potential techniques in the field of MBT in a variety of applications have evolved as a result of the work given in this paper. Work needs to be done in the area of automatic test case execution for testing PLC software and IEC 61131-3 applications. As indicated in [58] [59], current techniques for platforms of PLC still lack full support. Approaches that combine automatic execution into existing techniques or new methods aimed at PLC platforms are rare. When it comes to computer science-based methodologies like regression testing and change effect analysis, this is also true. In terms of traceability-based CIA, these techniques are hampered by the lack of defined models for functional specification definition in production automation. This is most likely because of the unpredictability of investment returns (personnel training, software tools, work throughout the design phase is required, etc.) and return (better software quality and better efficiency, etc.), however, further investigations are required to identify the root causes of this problem. The IEC 61131-3 programming standard and the languages it specifies, structure, and aspects of execution appear to be impeding the use of several ideas from computer science in dependency-based CIA (real-time, cyclic) [60] [61]. Further research on the usability and benefits of these technologies in the context of many applications is still pending. Defining acceptable fault models and producing test cases to test the reaction to faults is still a new subject in many applications, with much of the research done so far in other disciplines [62]. It is common practice to utilize FI to verify the reliability of integrated circuits by testing for hardware flaws. Automotive and aerospace industries employ accessible models to introduce errors at the model level during the development process. Production automation on the application level has been underutilized in areas where models are rare, such as FI approaches, even though reliability and dependability are of enormous relevance in this industry [63]. Several HiL techniques in the neighboring domain of industrial automation have been found throughout the article [64]. HiL techniques that focus on production automation, on the other hand, are quite rare. We've looked into how to generate test cases from models created with engineering

modeling languages like UML and UML-based UML, as well as ways for doing so. Case studies and comprehensive surveys on the acceptability and applicability of these techniques, as well as the production of test cases from SysML techniques in many applications, are still a work in progress. For the modification of well-formal models based on known models for the development of test cases, methods and test selection criteria are available in varying forms. Evaluation of industrial could aid in determining the approaches' industrial applicability for the domain of many applications. The implementation of the plant model should comprise not just the source code but also, at a minimum, the implementation model[65][66].

MBT methodologies are discussed in this study in light of current issues in the field of various applications. Based on [2], these issues have been categorized as the production of test cases that are appropriate to the PLC software, the reduction of testing time and effort through testing of regression, the integration of hardware impacts, and the creation of test cases using industry-accepted models. To further classify the different methodologies, models and requirements are used as a fundamental for testing, test choice criteria, and classification criteria are introduced, the system being evaluated and the results of the evaluation, testbed, and executable test cases. In conclusion, additional work needs to be done, particularly in terms of creating test case development based on user-friendly notations, testing the system's behavior to regression, and faults testing.

6. Conclusions

Model-based testing (MBT) is gaining traction in the software testing community as a way to use a model of the behavior of the system under test to support one or more software testing tasks. Simultaneously, there has been a huge number of MBT strategies proposed in the literature. There is a discrepancy between the research on MBT and what MBT tools support, according to our experience, i.e., Most MBT studies don't result in MBT tools that can be used. A systematic evaluation of existing MBT tools, with a focus on state-based testing tools, has been performed as a first step to better comprehend this knowledge gap. We found 27 tools and compared nine of them using a rigorous process. As part of our evaluation, we look at adequacy/coverage criteria based on models and scripts, as well as assistance with associated testing tasks (such as building a test model) and assistance with building a test scaffolding (stub, oracle, driver). Except for simple criteria like transition/ state adequacy criterion and test case development, the results reveal that support for those comparison criteria varies a lot from tool to tool. There is a lot of space for improvement in the area of test scaffolding and test data selection criteria, according to our opinion. The paper's key contributions are three-fold. In the first place, systematic literature review principles are being employed to research state-based MBT technologies for the first time. Second, the methodical approach allowed us to explicitly specify criteria for comparison, which will complement what is currently available in the literature comparing similar instruments. Please keep in mind that our method was standardized, which should allow it to be easily replicated and expanded upon. Third, the results clearly show how the instruments and which criteria they most frequently fail to meet.

References

[1]. Liu, Y., Li, Y., Lin, S. W., & Yan, Q. (2020, November). ModCon: A model-based testing platform for smart contracts. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1601-1605).

[2]. Camilli, M., & Russo, B. (2020, September). Model-based testing under parametric variability of uncertain beliefs. In International Conference on Software Engineering and Formal Methods (pp. 175-192). Springer, Cham.

[3]. Kanter, G., & Vain, J. (2020). Model-based testing of autonomous robots using TestIt. Journal of Reliable Intelligent Environments, 6(1), 15-30.

[4]. Canny, A., Palanque, P., & Navarre, D. (2020, October). Model-Based Testing of GUI Applications Featuring Dynamic Instantiation of Widgets. In 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 95-104). IEEE.

[5]. Petry, K. L., OliveiraJr, E., & Zorzo, A. F. (2020). Model-based testing of software product lines: Mapping study and research roadmap. Journal of Systems and Software, 167, 110608.

[6]. Tanabe, K., Tanabe, Y., & Hagiya, M. (2020, August). Model-based testing for MQTT applications. In Joint Conference on Knowledge-Based Software Engineering (pp. 47-59). Springer, Cham.

[7]. Silistre, A., Kilincceker, O., Belli, F., Challenger, M., & Kardas, G. (2020, September). Community detection in model-based testing to address scalability: Study design. In 2020 15th Conference on Computer Science and Information Systems (FedCSIS) (pp. 657-660). IEEE.

[8]. Lambers, L., Schneider, S., & Weisgut, M. (2020, October). Model-based testing of read-only graph queries. In 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 24-34). IEEE.

[9]. Akpinar, P., Aktas, M. S., Keles, A. B., Balaman, Y., Guler, Z. O., & Kalipsiz, O. (2020, June). Web application testing with model-based testing method: case study. In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE) (pp. 1-6). IEEE.

[10]. Panizo, L., Díaz, A., & García, B. (2020). Model-based testing of apps in real network scenarios. International Journal on Software Tools for Technology Transfer, 22(2), 105-114.

[11]. Sabbaghi, A. (2021). A Classification Framework of Test Models in Model-based Testing. Future Generation of Communication and Internet of Things, 1(2), 1-14.

[12]. Ahmad, T., Iqbal, J., Ashraf, A., Truscan, D., & Porres, I. (2019). Model-based testing using UML activity diagrams: A systematic mapping study. Computer Science Review, 33, 98-112.

[13]. Gurbuz, H. G., & Tekinerdogan, B. (2018). Model-based testing for software safety: a systematic mapping study. Software Quality Journal, 26(4), 1327-1372.

[14]. Gebizli, C. Ş., & Sözer, H. (2017). Automated refinement of models for model-based testing using exploratory testing. Software Quality Journal, 25(3), 979-1005.

[15]. Belli, F., Endo, A. T., Linschulte, M., & Simao, A. (2014). A holistic approach to model-based testing of Web service compositions. Software: Practice and Experience, 44(2), 201-234.

[16]. Marinescu, R., Seceleanu, C., Le Guen, H., & Pettersson, P. (2015). A research overview of tool-supported model-based testing of requirements-based designs. Advances in Computers, 98, 89-140.

[17]. Aerts, A., Reniers, M., & Mousavi, M. R. (2017). Model-based testing of cyber-physical systems. In Cyber-Physical Systems (pp. 287-304). Academic Press.

[18]. Coutinho, A. E. V. B., Cartaxo, E. G., & Machado, P. D. D. L. (2016). Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing. Software Quality Journal, 24(2), 407-445.

[19]. Eduard P Enoiu, Adnan Causevic, Daniel Sundmark, and Paul Pettersson. 2016. A controlled experiment in testing of safety-critical embedded software. In 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE, 1–11.

[20]. Christoph Schulze, Dharmalingam Ganesan, Mikael Lindvall, Rance Cleaveland, and Daniel Goldman. 2014. Assessing Model-Based Testing: An Empirical Study Conducted in Industry. In Companion Proceedings of the 36th International Conference on Software Engineering. Association for Computing Machinery, New York, NY, USA.

[21]. Alexander Pretschner, Wolfgang Prenninger, Stefan Wagner, Christian Kühnel, Martin Baumgartner, Bernd Sostawa, Rüdiger Zölch, and Thomas Stauner. 2005. One evaluation of model-based testing and its automation. In Proceedings of the 27th international conference on Software engineering. 392–401.

[22]. Damiani, F., Faitelson, D., Gladisch, C., & Tyszberowicz, S. (2017). A novel model-based testing approach for software product lines. Software & Systems Modeling, 16(4), 1223-1251.

[23]. Ali, A., Maghawry, H. A., & Badr, N. (2021, December). Model-Based Test Case Generation Approach for Mobile Applications Load Testing using OCL Enhanced Activity Diagrams. In 2021 Tenth International Conference on Intelligent Computing and Information Systems (ICICIS) (pp. 493-499). IEEE.

[24]. Pilarowski, G., Marquez, C., Rubio, L., Peng, J., Martinez, J., Black, D., ... & Havlir, D. V. (2021). Field performance and public health response using the BinaxNOWTM rapid severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) antigen detection assay during community-based testing. Clinical Infectious Diseases, 73(9), e3098-e3101.

[25]. Bonifacio, A. L., & Moura, A. V. (2021). Testing Asynchronous Reactive Systems: Beyond the ioco framework. CLEI ELECTRONIC JOURNAL, 24(2).

[26]. Mishra, A., & Macgregor, S. (2015). VEGAS2: software for more flexible gene-based testing. Twin Research and Human Genetics, 18(1), 86-91.

[27]. Lambert, N., Wilcox, A., Zhang, H., Pister, K. S., & Calandra, R. (2021, December). Learning accurate long-term dynamics for model-based reinforcement learning. In 2021 60th IEEE Conference on Decision and Control (CDC) (pp. 2880-2887). IEEE.

[28]. Du, Z., Pandey, A., Bai, Y., Fitzpatrick, M. C., Chinazzi, M., y Piontti, A. P., ... & Meyers, L. A. (2021). Comparative cost-effectiveness of SARS-CoV-2 testing strategies in the USA: a modeling study. The Lancet Public Health, 6(3), e184-e191.

[29]. Felderer, M., & Schieferdecker, I. (2014). A taxonomy of risk-based testing. International Journal on Software Tools for Technology Transfer, 16(5), 559-568.

[30]. Uzun, B., & Tekinerdogan, B. (2018). Model-driven architecture-based testing: A systematic literature review. Information and Software Technology, 102, 30-48.

[31]. Evans, S., Agnew, E., Vynnycky, E., Stimson, J., Bhattacharya, A., Rooney, C., & Robotham, J. (2021). The impact of testing and infection prevention and control strategies on within-hospital transmission dynamics of COVID-19 in English hospitals. Philosophical Transactions of the Royal Society B, 376(1829), 20200268.

[32]. Aichernig, B. K., Brandl, H., Jöbstl, E., Krenn, W., Schlick, R., & Tiran, S. (2015). Killing strategies for model-based mutation testing. Software Testing, Verification and Reliability, 25(8), 716-748.

[33]. Subramanian, R., He, Q., & Pascual, M. (2021). Quantifying asymptomatic infection and transmission of COVID-19 in New York City using observed cases, serology, and testing capacity. Proceedings of the National Academy of Sciences, 118(9).

[34]. Wang, Y., Yi, Y. F., Li, C. H., & Han, J. Q. (2021). Anisotropic fracture and energy characteristics of a Tibet marble exposed to multi-level constant-amplitude (MLCA) cyclic loads: a lab-scale testing. Engineering fracture mechanics, 244, 107550.

[35]. Chen, Y., Probert, R.L. and Ural, H. (2009) Regression Test Suite Reduction Based on SDL Models of System Requirements. Journal of Software Maintenance and Evolution: Research and Practice, 21, 379-405. http://dx.doi.org/10.1002/smr.415.

[36]. Aichernig, B. K., Jöbstl, E., & Tiran, S. (2015). Model-based mutation testing via symbolic refinement checking. Science of Computer Programming, 97, 383-404.

[37]. Caliebe, P., Herpel, T. and German, R. (2012) Dependency-Based Test Case Selection and Prioritization in Embedded Systems. 2012 IEEE 5th International Conference on Software Testing, Verification and Validation, Montreal, 17-21 April 2012, 731-735. http://dx.doi.org/10.1109/ICST.2012.164.

[38]. Powell, D., Arlat, J., Chu, H.N., Ingrand, F. and Killijian, M. (2012) Testing the Input Timing Robustness of Real Time Control Software for Autonomous Systems. 2012 9th European Dependable Computing Conference, Sibiu, 8-11 May 2012, 73-83. http://dx.doi.org/10.1109/EDCC.2012.16.

[39]. Korel, B., Tahat, L. and Vaysburg, B. (2002) Model-Based Regression Test Reduction Using Dependence Analysis. 2002 International Conference on Software Maintenance, Montreal, 3-6 October 2002, 214-223. http://dx.doi.org/10.1109/icsm.2002.1167768.

[40]. Maropoulos, P. G., Vichare, P., Martin, O., Muelaner, J., Summers, M. D., & Kayani, A. (2011). Early design verification of complex assembly variability using a Hybrid–Model Based and Physical Testing–Methodology. CIRP annals, 60(1), 207-210.

[41]. Farooq, Q., Iqbal, M.Z.Z., Malik, Z.I. and Nadeem, A. (2007) An Approach for Selective State Machine Based Regression Testing. Proceedings of the 3rd International Workshop on Advances in Model-Based Testing (A-MOST'07), ACM Press, New York, 44-52. http://dx.doi.org/10.1145/1291535.1291540.

[42]. Caliebe, P., Herpel, T. and German, R. (2012) Dependency-Based Test Case Selection and Prioritization in Embedded Systems. 2012 IEEE 5th International Conference on Software Testing, Verification and Validation, Montreal, 17-21 April 2012, 731-735. http://dx.doi.org/10.1109/ICST.2012.164.

[43]. Leung, H. and White, L. (1990) A Study of Integration Testing and Software Regression at the Integration Level. 1990 International Conference on Software Maintenance, San Diego, 26-29 November 1990, 290-301. http://dx.doi.org/10.1109/icsm.1990.13137.

[44]. Dionysiou, K., Bolbot, V., & Theotokatos, G. (2022). A functional model-based approach for ship systems safety and reliability analysis: Application to a cruise ship lubricating oil system. Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment, 236(1), 228-244.

[45]. Mohalik, S., Gadkari, A. A., Yeolekar, A., Shashidhar, K. C., & Ramesh, S. (2014). Automatic test case generation from Simulink/Stateflow models using model checking. Software Testing, Verification and Reliability, 24(2), 155-180.

[46]. Kaminski, G., Ammann, P., & Offutt, J. (2013). Improving logic-based testing. Journal of Systems and Software, 86(8), 2002-2012.

[47]. Wang, X., & Hong, Y. (2018). Characteristic function based testing for conditional independence: A nonparametric regression approach. Econometric Theory, 34(4), 815-849.

[48]. Wang, C., Xu, G., & Shang, Z. (2018). A two-stage approach to differentiating normal and aberrant behavior in computer based testing. Psychometrika, 83(1), 223-254.

[49]. Sari, Y. I., Utomo, D. H., & Astina, I. K. (2021). The Effect of Problem Based Learning on Problem Solving and Scientific Writing Skills. International Journal of Instruction, 14(2), 11-26.

[50]. Aparow, V. R., Hong, C. J., Weun, N. Y., Huei, C. C., Yen, T. K., Hong, L. C., ... & Wen, K. K. (2021, December). Scenario based Simulation Testing of Autonomous Vehicle using Malaysian Road. In 2021 5th International Conference on Vision, Image and Signal Processing (ICVISP) (pp. 33-38). IEEE.

[51]. Butler-Laporte, G., Lawandi, A., Schiller, I., Yao, M., Dendukuri, N., McDonald, E. G., & Lee, T. C. (2021). Comparison of saliva and nasopharyngeal swab nucleic acid amplification testing for detection of SARS-CoV-2: a systematic review and meta-analysis. JAMA internal medicine, 181(3), 353-360.

[52]. Jarodzka, H., Janssen, N., Kirschner, P. A., & Erkens, G. (2015). Avoiding split attention in computer-based testing: Is neglecting additional information facilitative?. British journal of educational technology, 46(4), 803-817.

[53]. Lindner, M. A., Eitel, A., Barenthien, J., & Köller, O. (2021). An integrative study on learning and testing with multimedia: Effects on students' performance and metacognition. Learning and Instruction, 71, 101100.

[54]. Kazimov, T., Bayramova, T., & Malikova, N. (2021). Research of intelligent methods of software testing. System research and information technologies, (4), 42-52.

[55]. Zheng, H., Li, P., & Ma, G. (2021). Stability analysis of the middle soil pillar for asymmetric parallel tunnels by using model testing and numerical simulations. Tunneling and Underground Space Technology, 108, 103686.

[56]. Lüdecke, D., Ben-Shachar, M. S., Patil, I., Waggoner, P., & Makowski, D. (2021). performance: An R package for assessment, comparison and testing of statistical models. Journal of Open Source Software, 6(60).

[57]. Klinke, D. J. (2014). In silico model-based inference: A contemporary approach for hypothesis testing in network biology. Biotechnology Progress, 30(6), 1247-1261.

[58]. Sampath, S., & Bryce, R. C. (2012). Improving the effectiveness of test suite reduction for user-session-based testing of web applications. Information and Software Technology, 54(7), 724-738.

[59]. Rösch, S., Ulewicz, S., Provost, J., & Vogel-Heuser, B. (2015). Review of model-based testing approaches in production automation and adjacent domains-current challenges and research gaps. Journal of Software Engineering and Applications.

[60]. Shafique, M., & Labiche, Y. (2010). A systematic review of model based testing tool support.

[61]. Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. Software testing, verification and reliability, 22(5), 297-312.

[62]. Zander, J., Schieferdecker, I., & Mosterman, P. J. (Eds.). (2017). Model-based testing for embedded systems. CRC press.

[63]. Peleska, J. (2013). Industrial-strength model-based testing-state of the art and current challenges. arXiv preprint arXiv:1303.1006.

[64]. Amalfitano, D., Fasolino, A. R., Tramontana, P., Ta, B. D., & Memon, A. M. (2014). MobiGUITAR: Automated model-based testing of mobile apps. IEEE software, 32(5), 53-59.

[65]. Estévez, E. and Marcos, M. (2012) Model-Based Validation of Industrial Control Systems. IEEE Transactions on Industrial Informatics, 8, 302-310. http://dx.doi.org/10.1109/TII.2011.2174248

[66]. Petrenko, A., Dury, A., Ramesh, S. and Mohalik, S. (2013) A Method and Tool for Test Optimization for Automotive Controllers. 2013 IEEE 6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Luxembourg, 18-22 March 2013, 198-207. http://dx.doi.org/10.1109/ICSTW.2013.31